CS4032D - Computer Architecture

CUDA - Part II - Matrix Multiplication using CUDA, OpenMP and pthread

Team Members

- B181065CS - Anish Sharma
- B181103CS - Aman Singh Kadiyan
- B180369CS - Aishik Rana
- B180403CS - Tarun Kansal

Inputs

All the implementations use two randomly generated vectors of a certain size as input.
Multiple iterations are run with the size of vectors ranging from 1000, 2000, 4000 ... up till less than 100000000. This allows measuring time taken for a wide range of input sizes.

Output

Each implementation outputs a CSV file containing the results in the format

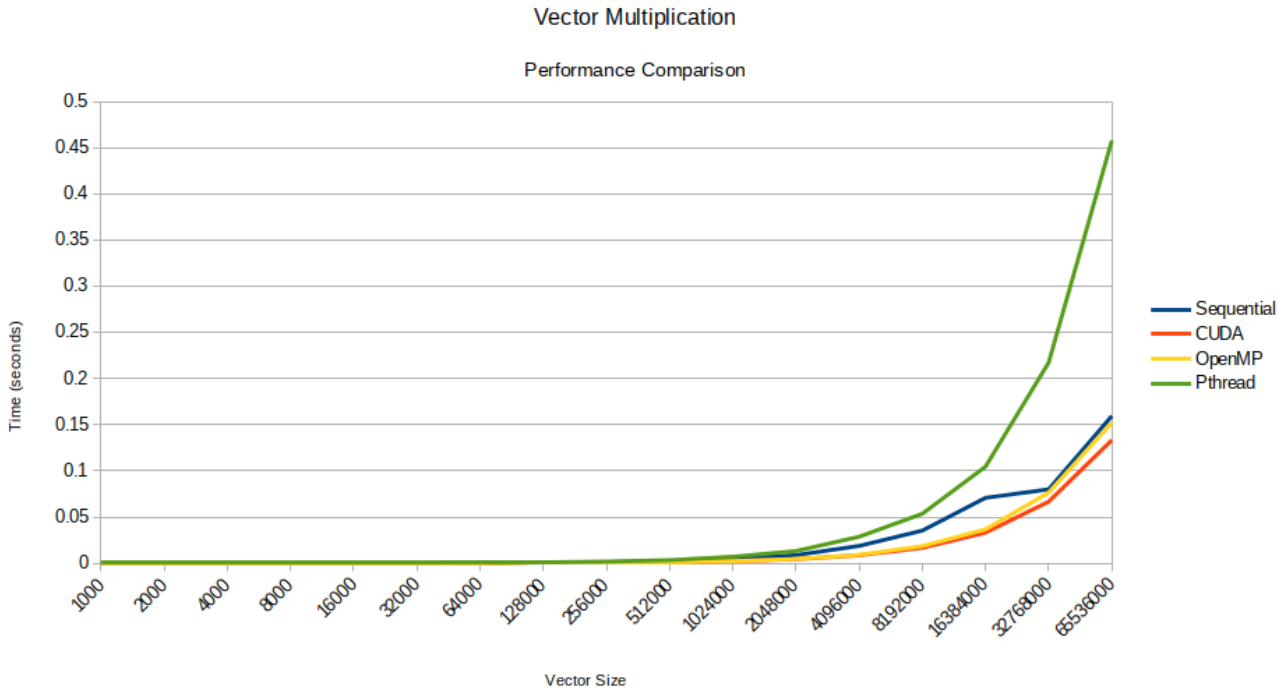
Array Size	Time Taken
1000	-
2000	-
4000	-

These results are combined into a single large CSV file containing results for all the implementations.

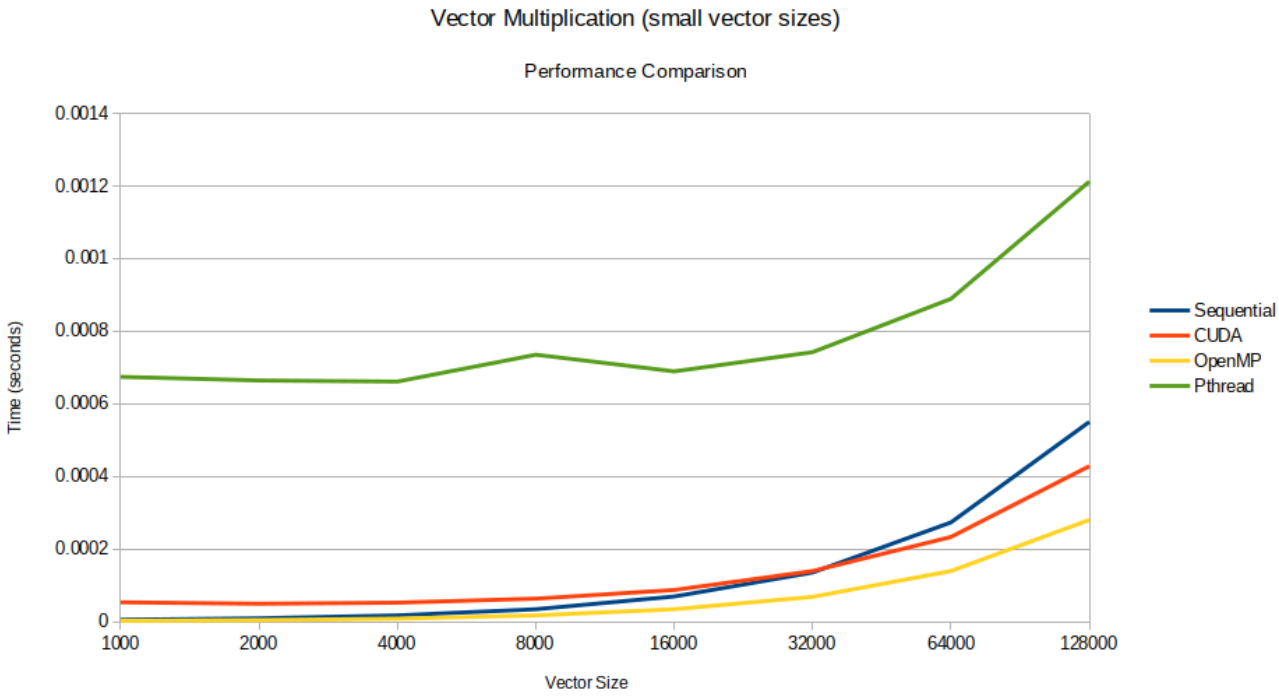
These results can then be compared using visualizations such as line charts.

Results

Overall



Small Vector sizes



Inferences

For large vector sizes, we can clearly see that CUDA and OpenMP perform better than Sequential operations. This is due to the fact that vector multiplication is easy to parallelized.

However for small vector sizes, we see that sequential operations perform a lot better than CUDA. This can be attributed to the overhead caused during

parallelization of operations. In case of CUDA, copying memory from host to device and vice versa.

OpenMP seems to perform consistently as good as sequential and better for large sizes. This could be due to optimizations in OpenMP for smaller operations as well as due to much less overhead compared to CUDA.

On the other hand, the pthread implementation seems to perform much worse than even sequential for all array sizes. This can be due to the fact that pthread creation, deletion etc. as well as context switching gives a much larger overhead. Also, pthread might not be able to utilize the CPU threads (hardware threads) effectively.

These all factors lead to pthreads not being feasible in this use case.